

SOFTWARE METAPAPER

The Matlab Radial Basis Function Toolbox

Scott A. Sarra

Marshall University, US

sarra@marshall.edu

Radial Basis Function (RBF) methods are important tools for scattered data interpolation and for the solution of Partial Differential Equations in complexly shaped domains. The most straight forward approach used to evaluate the methods involves solving a linear system which is typically poorly conditioned. The Matlab Radial Basis Function toolbox features a regularization method for the ill-conditioned system, extended precision floating point arithmetic, and symmetry exploitation for the purpose of reducing flop counts of the associated numerical linear algebra algorithms.

Keywords: Radial Basis Functions; numerical partial differential equations; extended precision; Matlab; object oriented programming

(1) Overview**Introduction**

Radial Basis Function (RBF) methods are important tools for scattered data interpolation and for the solution of PDEs in complexly shaped domains. The most straight forward approach that is used to evaluate the method while incorporating the “standard basis functions” involves solving a linear system which is typically poorly conditioned. Two variations of a method, dubbed the RBF-QR approach, use a different basis that spans the same space as the standard basis but which in some cases results in a better conditioned linear system. Software packages which implement the two RBF-QR approaches are freely available ([11] and [13]).

Extended precision floating point arithmetic can be used to accurately evaluate the ill-conditioned problem in the standard basis. This approach has been used and implemented in several different software environments that include: Mathematica [7], the Matlab Symbolic Toolbox [21], C++ [16], and Fortran [5]. The extended precision approach is attractive because it retains one of the great strengths of the RBF method – simplicity. Whereas the RBF-QR approaches, as can be ascertained by browsing the software that implements the methods, is far more complex. A software package that implements the RBF method in extended precision is not currently available. A detailed comparison of the RBF-QR and extended precision with the standard basis approaches is made in [20].

In this work a Matlab [12] toolbox is described that features a regularization method for the ill-conditioned linear system, extended precision floating point arithmetic, and symmetry exploitation for the purpose of reducing flop counts. The toolbox is called the Matlab Radial Basis

Function Toolbox (MRBFT). The toolbox uses an object oriented approach to organize its functionality via three main classes. Static methods in the class `rbfx` are used to implement functionality associated with RBF methods in general, while class methods are used to implement methods in subclasses of `rbfx` which apply to a particular RBF. The superclass `rbfx` has abstract methods which every subclass must implement. These include a definition of the RBF itself as well as a variety of derivative operators applied to the RBF. The complete list of abstract methods is as follows:

```
methods(Abstract = true)
v = rbf(obj, r, s); % RBF definition
d = D1(obj, r, s, x); % first derivative wrt x
d = D2(obj, r, s, x); % second derivative wrt x
d = D3(obj, r, s, x); % third derivative wrt x
d = D4(obj, r, s, x); % fourth derivative wrt x
d = G(obj, r, s, x, y); % Gradient
d = L(obj, r, s); % Laplacian
d = B(obj, r, s, x, y); % Biharmonic operator
d = D12(obj, r, s, x, y); % mixed partial derivative
d = D22(obj, r, s, x, y); % mixed partial derivative
end
```

The `rbfx` class is subclassed by the `gax` and `iqx` classes which implement methods that are respectively particular to the Inverse Quadratic and Gaussian RBF. The class `rbfCenters` implements methods associated with scattered center locations and the class `rbfCentro` implements methods associated with algorithms that accurately and efficiently operate on structured matrices.

Radial Basis Function Methods

RBF interpolation uses a set of N distinct points $X = \{x_1^c, \dots, x_N^c\}$ in \mathcal{R}^d called centers. No restrictions are placed on the shape of problem domains or on the location of the centers. A RBF

$$\phi_k(x) = \phi(\|x - x_k^c\|_2, \varepsilon_k), \quad x, x_k^c \in \mathcal{R}^d \quad (1)$$

is an infinitely differentiable (compactly supported and global RBFs without a shape parameter and with less smoothness exist but are not considered here) function of one variable $r = \|x - x_k^c\|_2$ that is centered at x_k^c and that contains a free parameter ε called the shape parameter. The RBF interpolant assumes the form

$$I_N f(x) = \sum_{k=1}^N a_k \phi(\|x - x_k^c\|_2, \varepsilon_k) \quad (2)$$

where a is a vector of expansion coefficients. The Gaussian (GA) RBF

$$\phi(r) = e^{-\varepsilon^2 r^2} \quad (3)$$

and the inverse quadratic (IQ) RBF

$$\phi(r) = \frac{1}{1 + \varepsilon^2 r^2} \quad (4)$$

are representative members of the class of global, infinitely differentiable RBFs containing a shape parameter that interpolate with exponential accuracy. The two RBFs and their various derivatives are defined respectively in the classes `gax` and `iqx`. A particular RBF, for example the GA, is instantiated via

```
>> phi = iqx(); % phi is an instance of the
inverse quadratic RBF class
```

Many other RBFs exist and may be added by the user to the toolbox by extending `rbfx` and using `gax` and `iqx` as examples.

The entire list of functions associated with an object is available as follows:

```
> methods(iqx)

Methods for class iqx:

B          D2          D4          iqx
D1         D22         G           rbf
D12        D3          L

Static methods:

distanceMatrix1d  distanceMatrix3d  solve
distanceMatrix2d  dm                variableShape
```

First the methods that the class must define are listed and then the static methods inherited from the superclass are listed.

The RBF expansion coefficients are determined by enforcing the interpolation conditions

$$I_N f(x_k^c) = f(x_k^c), \quad k = 1, 2, \dots, N \quad (5)$$

which result in a $N \times N$ linear system

$$Ba = f. \quad (6)$$

The matrix B with entries

$$b_{jk} = \phi(\|x_j^c - x_k^c\|_2, \varepsilon_k), \quad j, k = 1, \dots, N \quad (7)$$

is called the system matrix. The solve method which is a static method of the `rbfx` class can be used to find the expansion coefficients. Static methods can be called in two ways. Either through the class

```
>> a = iqx.solve(B, f); % or a = rbfx.solve(B, f);
```

or from instances of the class

```
>> a = phi.solve(B, f);
```

Matlab is optimized for operations involving matrices and vectors. The process of revising loop-based, scalar-oriented code to use Matlab matrix and vector operations is called vectorization. At every opportunity, functions have been vectorized so that they execute as efficiently as possible.

The functions that define RBFs and their derivative matrices take distance matrices as their arguments. Taking for example 2d where $x^c = (x_1^c, x_2^c)$ the signed distance matrices

$$(rx)_{jk} = (x_1^c)_j - (x_1^c)_k \quad \text{and} \quad (ry)_{jk} = (x_2^c)_j - (x_2^c)_k \quad \text{where} \quad j, k = 1, \dots, N \quad (8)$$

respectively contain the signed distance between the x and y coordinates of centers j and k . The distance matrix

$$r_{jk} = \sqrt{(rx)_{jk}^2 + (ry)_{jk}^2}, \quad j, k = 1, \dots, N \quad (9)$$

stores the distance between centers j and k . With the x and y coordinates of the centers located in arrays xc and yc the distance matrices are formed via

```
>> [r, rx, ry] = rbfx.distanceMatrix2d(xc, yc);
```

and the RBF system matrix is constructed as

```
>> B = phi.rbf(r, s);
```

The evaluation of the interpolant (2) at M points x_j can be accomplished by multiplying the expansion coefficients by the $M \times N$ evaluation matrix H that has entries

$$h_{jk} = \phi(\|x_j - x_k^c\|_2, \varepsilon_k), \quad j = 1, \dots, M \quad \text{and} \quad k = 1, \dots, N \quad (10)$$

Continuing the 2d example, consider evaluation points $x = (x_1, x_2)$. The distance matrices containing the distances between the centers and evaluation points

$$(rxe)_{jk} = (x_1)_j - (x_1^c)_k \quad \text{and} \quad (rye)_{jk} = (x_2)_j - (x_2^c)_k \quad \text{where} \quad j = 1, \dots, M \quad \text{and} \quad k = 1, \dots, N \quad (11)$$

respectively contain the signed distances between the x and y coordinates of evaluation point j and center k . Then

$$re_{jk} = \sqrt{(rx_e)_{jk}^2 + (ry_e)_{jk}^2}, \quad (12)$$

$$j = 1, \dots, M \text{ and } k = 1, \dots, N.$$

With the x and y coordinates located in arrays x and y the distance matrices are formed via

```
>> [re, rx_e, ry_e] = rbfx.distanceMatrix2d(xc, yc, x, y);
and the evaluation matrix is constructed via
>> H = phi.rbf(re, s);
```

The interpolant is then evaluated as

```
>> fa = H*a;
```

Derivatives are approximated by differentiating the RBF interpolant as

$$\mathcal{D}(I_N f(x)) = \sum_{k=1}^N a_k \mathcal{D}\phi(\|x - x_k^c\|_2, \varepsilon_k) \quad (13)$$

where \mathcal{D} is a linear differential operator. The operator \mathcal{D} may be a single differential operator or a linear differential operator such as the Laplacian. Evaluating (13) at the centers X can be accomplished by multiplying the expansion coefficients by the evaluation matrix H_D with entries

$$h_{jk} = \mathcal{D}\phi(\|x_j^c - x_k^c\|_2, \varepsilon_k), \quad j, k = 1, \dots, N. \quad (14)$$

That is, $\mathcal{D}f \approx H_D a$. For example, to approximate the first derivative with respect to x of a function of two variables using the MRBFT

```
>> Hd = phi.D1(r, s, rx);
>> fa = Hd*a;
```

Alternatively, derivatives can be approximated by multiplying the grid function values $\{f(x_k^c)\}_{k=1}^N$ by the differentiation matrix $D = H_D B^{-1}$ since

$$\mathcal{D}f \approx H_D a = H_D (B^{-1} f) = (H_D B^{-1}) f. \quad (15)$$

This is accomplished as

```
>> D = phi.dm(B, Hd);
>> fa = D*f;
```

The shape parameter ε_k may take on different values at each center x_k^c (or equivalently in each column of the system or evaluation matrix). Such an approach is called a variable shape parameter strategy. Numerical evidence exists [8, 10, 22] that indicates that the use of a variable shape parameter may improve the conditioning of the system matrix as well as improve accuracy. A drawback of variable shape parameter strategies is that they cause the RBF system matrix to be non-symmetric. Reference [8] suggested the exponentially varying shape parameter

$$\varepsilon_j = \left[\varepsilon_{\min}^2 \left(\frac{\varepsilon_{\max}^2}{\varepsilon_{\min}^2} \right)^{\frac{j-1}{N-1}} \right]^{\frac{1}{2}} \quad j = 1, \dots, N, \quad (16)$$

and well as the linearly varying shape parameter

$$\varepsilon_j = \varepsilon_{\min} + \left(\frac{\varepsilon_{\max} - \varepsilon_{\min}}{N-1} \right) j \quad j = 0, 1, \dots, N-1. \quad (17)$$

Reference [22] gives examples of the benefits of using a random variable shape parameter

$$\varepsilon_j = \varepsilon_{\min} + (\varepsilon_{\max} - \varepsilon_{\min}) \times \text{rand}(1, N) \quad (18)$$

All three variable shape strategies are available in the MRBFT via

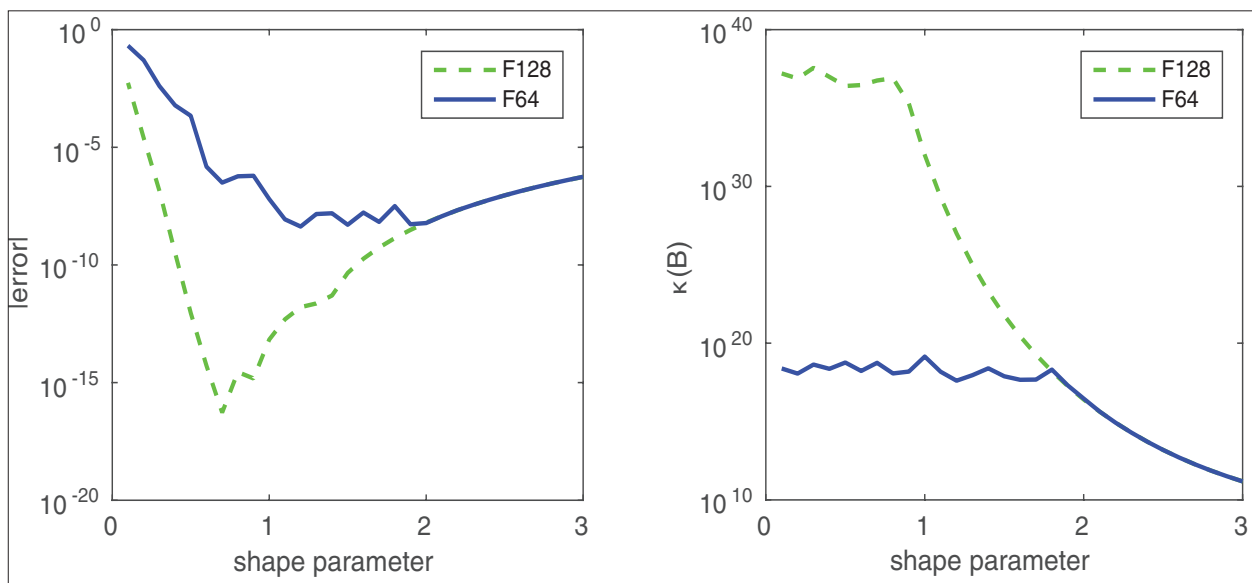


Figure 1: Interpolation of a smooth function using both double (F64) and quadruple (F128) precision. The script `condVaccuracy.m` produces the plots. Left: accuracy versus the shape parameter. Right: condition number of the system matrix versus the shape parameter.

```
>> Bs, Hs = phi.variableShape(sMin,sMax,opt,N,M);
```

Both equations (6) for the expansion coefficients and (15) for the differentiation matrix assume that the system matrix is invertible. Both the GA and IQ system matrices are symmetric positive definite (SPD) if a constant shape parameter is used and therefore they are invertible. While it is invertible, the system matrix is typically very poorly conditioned and it may cease to be numerically SPD and the standard algorithm to factorize a SPD matrix may fail. The eigenvalues of B satisfy $0 < \lambda_{min} = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_N = \lambda_{max}$ and the matrix condition number in the 2-norm is $\kappa(B) = \lambda_{min}/\lambda_{max}$. For a fixed set of centers, the shape parameter affects both the accuracy of the method and the conditioning of the system matrix. The RBF method is most accurate for smaller values of the shape parameter where the system matrix is ill-conditioned. **Figure 1** illustrates a typical result. For fixed N , the error is reduced as the shape parameter is made smaller until the condition number of the system matrix reaches $\mathcal{O}(10^{18})$ in double precision and $\mathcal{O}(10^{36})$ in quadruple precision. A regularization technique to mitigate the effects of the ill-conditioning is discussed in the next section.

Recent monographs [2, 4, 21, 23] on RBF methods can be consulted for more information on RBF methods in general.

Regularization

Reference [17] demonstrated that a simple regularization technique can mitigate the effects of the poor conditioning of RBF system matrices and in most cases ensure that the RBF system matrix remains numerically SPD so that Cholesky factorization can be used. Instead of solving the system

$$Ba = f \tag{19}$$

the regularized system

$$Cy = f \tag{20}$$

where $C = B + \mu I$ is solved. The parameter μ is a small positive constant called the regularization parameter and I is the identity matrix. The technique is called the method of diagonal increments (MDI) and its first use dates back to the 1940's [15]. Matrix C is better conditioned than B as

$$\kappa(C) = \frac{\lambda_{max} + \mu}{\lambda_{min} + \mu} < \kappa(B) = \frac{\lambda_{max}}{\lambda_{min}}. \tag{21}$$

For small μ , $(B + \mu I)^{-1}$ is close to B^{-1} and MDI simply replaces B with $(B + \mu I)$ in computing the solution of a system. Equation

$$B^{-1} - (B + \mu I)^{-1} = \mu^2 B^{-1} (I + B^{-1} / \mu) B \tag{22}$$

quantifies how close that $(B + \mu I)^{-1}$ and B^{-1} are [6]. For very small μ the difference is negligible. A good choice of the parameter is $\mu = 5 \epsilon_M$ where ϵ_M is machine epsilon in the floating point number system being used.

All MRBFT functions that involve the factorization of a SPD matrix take two optional arguments: a regularization parameter μ and a logical variable **safe**. The default value of μ is $5e-15$ which is the suggested value for double precision. The standard linear equation solver in Matlab is the `mldivide` function which may be evoked via the backslash operator. If a matrix is symmetric, Cholesky factorization is attempted. If Cholesky factorization fails, then the matrix is factorized with LU factorization. The default value of `safe` is true which causes all MRBFT routines to use the `mldivide` function. Setting `safe` to false forces the routines to use Cholesky factorization. The danger in directly calling Cholesky factorization is that if the regularization parameter is not large enough a matrix may fail to be numerically SPD and Cholesky factorization will fail due to taking the square root of a negative number if the matrix is severely ill-conditioned.

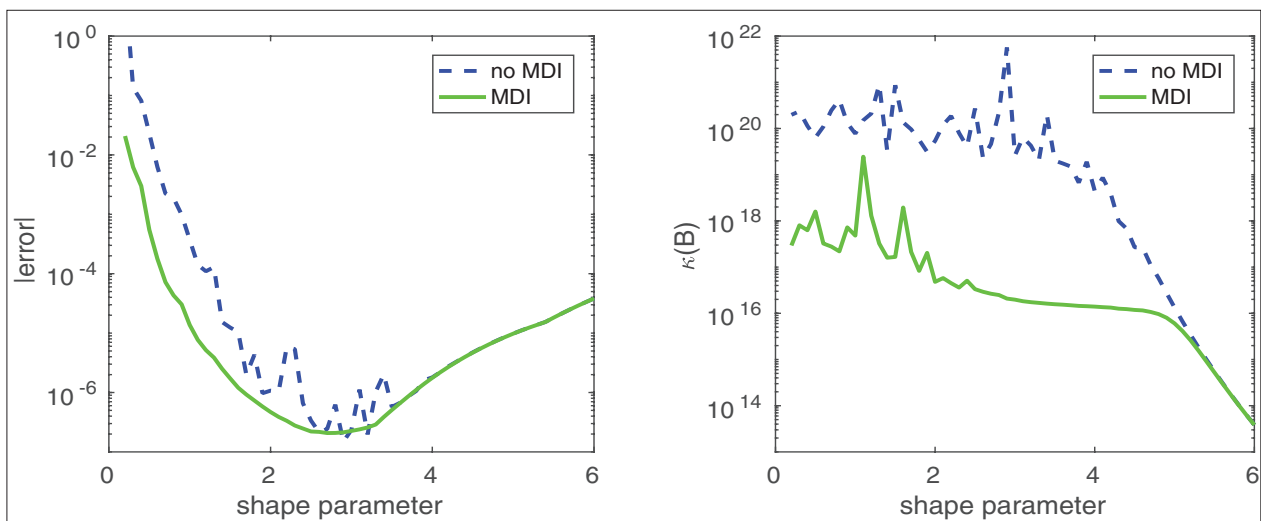


Figure 2: Interpolation with (green solid) and without (blue dashed) regularization. The script `mdiRegularization.m` produces the plots. Left: accuracy versus the shape parameter. Right: system matrix condition versus the shape parameter.

Figure 2 shows the results of a 2d interpolation problem with scattered centers with and without MDI regularization in double precision. Both the accuracy and condition number of the two approaches are virtually the same when the condition number of the system matrix can be accurately calculated in double precision which is for $\kappa(B) \approx \mathcal{O}(10^{16})$ and smaller. When the condition number of the unregularized method reaches beyond that threshold the regularization keeps the condition number in the approximately $\mathcal{O}(10^{16})$ range and the regularized solution is approximately two decimal places more accurate in this example.

Extended precision

The MRBFT uses the Multiprecision Computing Toolbox for Matlab (MCT) [1] for its extended precision functionality. The MCT enables extended precision data types to be seamlessly used in place of the standard double type. As a result, existing Matlab programs can be converted to run with arbitrary precision with minimal changes. IEEE 754-2008 compliant quadruple precision is supported and the MCT is highly optimized for this case. Note that the MRBFT is in no way dependent on the MCT. The installation of the MCT is not necessary. However, without the MCT, the MRBFT is limited to double precision.

The following code that calculates the RBF expansion coefficients gives an example of how to change double precision computations to extended precision:

```
1 phi = iqx();
2 mp.Digits(34);      % digits of decimal precision
3 N = mp('30');      % replace with N = 30 to
                    % convert to double precision
4 xc = linspace(0,1,N);
5 r = phi.distanceMatrix1d(xc);
6 B = phi.rbf(r,2.5);
7 f = sin(xc);
8 a = phi.solve(B,f,0);
```

The only coding difference in the above code between double and extended precision is on lines 2 and 3. Once the number of digits is specified and N is changed

to a mp object, all other operations involving the object are then done in extended precision. The script `interpBenchExtended.m` compares the execution speed of a 2d interpolation problem over a range of the shape parameter in both double and quadruple precision. In this particular example the quadruple precision calculation takes approximately 52 times longer to execute which is a typical result. In benchmarks, the MCT has been shown to be much more efficient in implementing extended precision than other available options. Additional comparisons of execution times can be found in reference [20].

Center locations

The second major class of the MRBFT is `rbfCenters`. The methods of the class are the following:

```
>> methods(rbfCenters)
Static methods:
Halton2d          circleCenters          squareCenters
Hammersley2d     circleUniformCenters
```

RBF methods place no restriction on the location of centers. However, randomly locating centers is unlikely to lead to good results. Theory dictates [2, 23] that centers should well-cover a domain in the sense that the centers are somewhat uniformly distributed with no large holes in their coverage and no centers clumped extremely close together. Centers located too close together hurt conditioning while large holes in the center coverage negatively affect convergence rates. Computational experience indicates that it is beneficial to locate centers more densely in boundary regions than in the interior of domains.

Quasirandom sequences [14] have become popular choices of centers for RBF methods. A quasirandom sequence is a n -tuples that fills n -space more uniformly than uncorrelated random points. Quasirandom sequences are also called low-discrepancy sequences. Halton points [24] are probably the most used quasirandom sequence in RBF methods due to the sequences

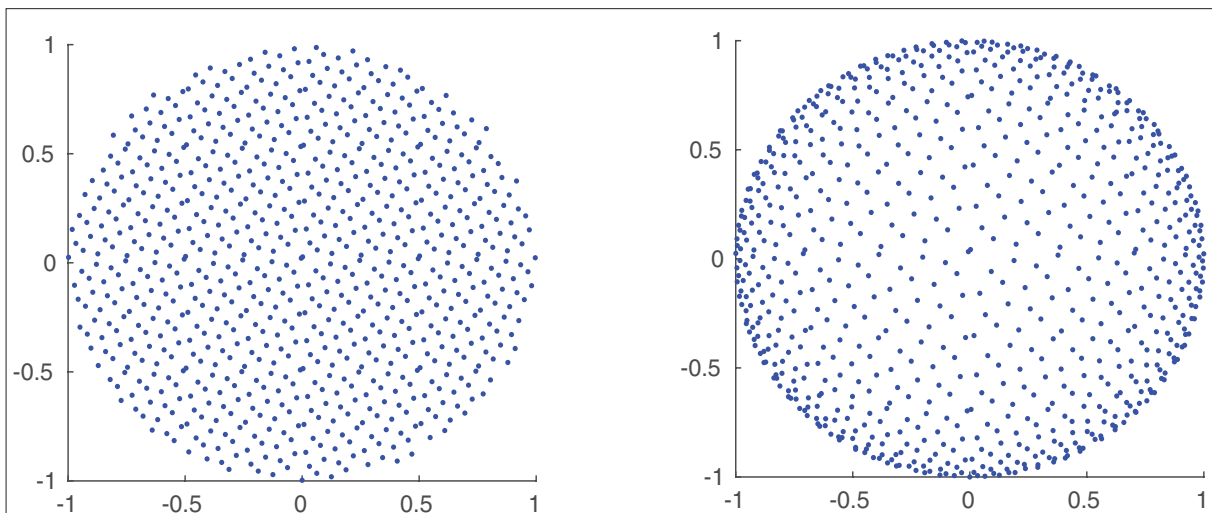


Figure 3: 1000 centers on the unit circle. Left: Hammersley points. Right: clustered Hammersley points.

being featured in the book [4]. However, it is our experience that the Hammersley points [24] provide a superior coverage in many cases. The MRBFT implements functions that produce both Halton and Hammersley center distribution for circular and square domains. The functions

```
>> [x, y] = rbfCenters.squareCenters(N,a,b,cluster,
    ch,plt); % square [a,b] x [a,b]
>> [x, y] = rbfCenters.circleCenters(N,cluster,
    ch,R,plt); % circle of radius R
```

provide the option to cluster the centers more densely around the boundaries of the domains. **Figure 3** shows a set of Hammersley points and a set of clustered Hammersley points on the unit circle. An example of how the MRBFT can be used to distribute boundary clustered quasirandom centers in a more complexly shaped domain can be found in the script `complexCentroCenters.m` in the examples directory of the MRBFT.

Symmetry

The third major class of the MRBFT is `rbfCentro` which takes advantage of symmetry to reduce the computational expense and storage requirements of RBF methods. The methods of the class are:

```
>> methods(rbfCentro)

Static methods:

centroCenters      centroDecomposeMatrix  hasSymmetry
centroCircle       centroEig                isCentro
centroCondition-   centroMult              isSkewCentro
Number
centroDM           fullCentroMatrix        solveCentro
```

A matrix B is centrosymmetric if $B = JBJ$ and is skew-centrosymmetric if $B = -JB_{21}$ where J , the contra-identity matrix, is a square matrix whose elements are all zero except those on its southwest-northeast diagonal which are all 1's. RBF system and differentiation matrices (details in [18]) are (skew) centrosymmetric if the signed distance matrices (11) are

(skew) centrosymmetric. Centrosymmetry does not depend on the location of centers, but rather on the distance between centers. Any center distribution in one dimension that is symmetrical about its mid-point causes the signed distance matrix to be skew-centrosymmetric. Center distributions in two dimensions that cause the signed distance matrix to be skew-centrosymmetric are easily generated in domains that are symmetrical with respect to either the x-axis, y-axis, or the origin, or that can be made so by a linear transformation or rotation.

Figure 4 shows two center distributions that result in centrosymmetric distance matrices. Centrosymmetry allows for significant flop count reductions in numerical linear algebra routines associated with RBF methods as well as reductions in computer memory requirements [18]. The efficiency gains when working with centrosymmetric matrices come from the fact that a $N \times N$ (where N is even and $M = N/2$) centrosymmetric matrix has the block structure

$$B = \begin{bmatrix} B_{11} & JB_{21}J \\ B_{21} & JB_{11}J \end{bmatrix} \tag{23}$$

where B_{11} , B_{21} , and J are $M \times M$. The block structure allows many linear algebra operations to be performed while only forming and operating on half of the matrix. Additionally, the matrix (22) and

$$D = \begin{bmatrix} L = B_{11} - JB_{21} & 0 \\ 0 & M = B_{11} + JB_{21} \end{bmatrix} \tag{24}$$

are orthogonally similar which allows for efficient eigenvalue calculation. Similar decompositions are possible for odd N as well.

The MRBFT functions for centrosymmetry include the following. Centrosymmetric center distributions in circular domains are produced by

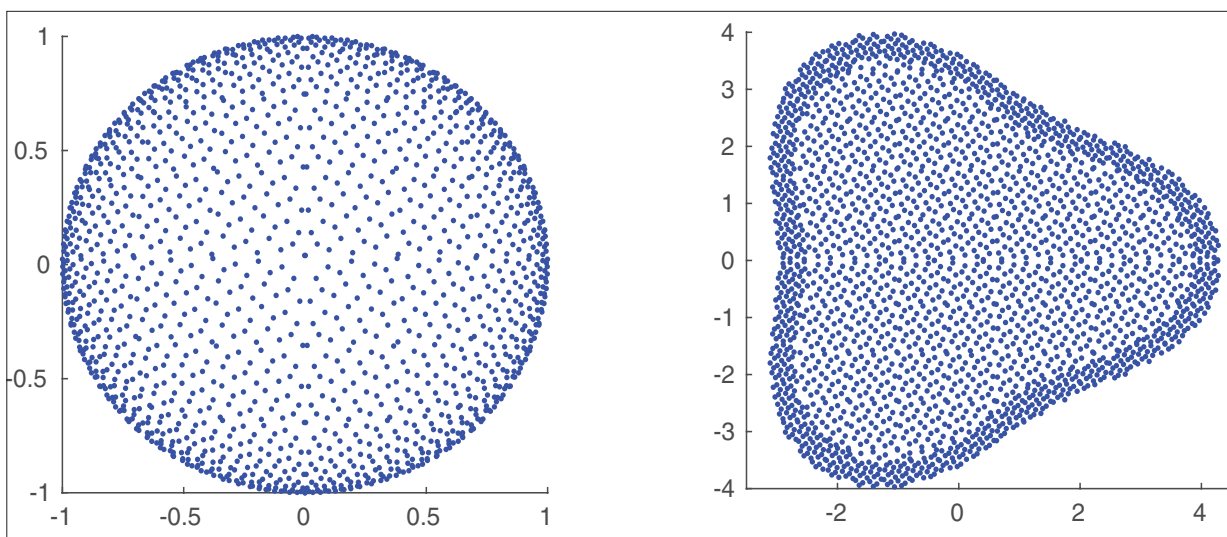


Figure 4: Center distributions that result in centrosymmetric distance matrices. Left: 2000 centers based on clustered Hammersley points that have been extended centro symmetrically about the x axis. Right: 2524 centers on a domain enclosed by the curve $f(\theta) = 3(\cos(3t) + \sqrt{4 - \sin(3t)^2})^{1/3}$ that result in centrosymmetric distance matrices.

```
>> [xc,yc] = rbfCentro.centroCircle
      (N,cluster,ch,R,plt);
```

and in domains with either x , y , or origin symmetry the function

```
>> [xc,yc] = rbfCentro.
      centroCenters(x,y,symType,plt);
```

uses half of the centers given as an argument to return a centrosymmetric center distribution.

Only half of the matrix needs to be formed and stored and then subsequently operated on in centrosymmetric linear algebra routines. The half-sized distance matrices are formed via calling the distanceMatrix functions with the arguments modified as follows:

```
>> r = rbfx.distanceMatrix1d(xc(1:N/2),xc);
>> [r, rx, ry] = rbfx.distanceMatrix2d(xc(1:N/2)
      ,yc(1:N/2),xc,yc);
```

The half sized distance matrices can be used to form half-sized system and derivative evaluation matrices as

```
>> B = phi.rbf(r,s);
>> H = phi.D2(r,s,rx);
```

Matrix condition numbers in the 2-norm are calculated via

```
>> [kappaB, kappaL, kappaM] = rbfCentro.
      centroConditionNumber(B,mu);
```

with a factor of four savings in the dominant term in the flop count compared to the standard algorithm. The RBF expansion coefficients for interpolation problems can be found from the half-sized system matrix as

```
>> a = rbfCentro.solveCentro(B,f,mu,safe);
```

Two quarter-sized matrices are factorized by a Cholesky factorization and the dominant term in the flop count is reduced by a factor of four from $\frac{1}{3}N^3$ to $\frac{1}{12}N^3$. In a similar manner, differentiation matrices are formed via

```
>> D = rbfCentro.centroDM(B,F,N,rho,mu,safe);
```

with a factor of four savings in computational effort. Centrosymmetric matrix-vector multiplication is asymptotically faster by a factor of two over the standard algorithm and can be accomplished via

```
>> [L,M] = rbfCentro.centroDecomposeMatrix(D,rho);
      % decompose into smaller matrices
>> fp = rbfCentro.centroMult(f,L,M,rho);
      % multiply and reconstruct solution
```

Other miscellaneous functions concerning centrosymmetry are

```
>> c = rbfCentro.hasSymmetry(B);
%tests a N x N matrix for (skew) centrosymmetry
```

which test matrices for symmetry and

```
>> D = rbfCentro.fullCentroMatrix(Dh,N,skew);
```

which constructs a full-size centrosymmetric matrix from a half-sized matrix.

Examples, tests, and benchmarks

The MRBFT distribution includes three directories that contains scripts that provide examples of using the functions in the toolbox. The **examples** directory

contains scripts that use the functions for typical tasks associated with the RBF method. The **tests** directory contains scripts that verify that the various functions are working as claimed. The **benchmarks** directory contains scripts that measure the execution time of the centrosymmetric algorithms versus the standard algorithms and that measure the execution time of algorithms in double precision versus extended precision. Below a brief description of each script is given. More detailed information is contained in the comments of each script. Extensive comments are also contained in the source code the the classes **rbfx**, **rbfCenters**, and **rbfCentro**.

The following example scripts come with the toolbox:

- **variableShapeInterp1d.m** Variable shape parameter versus constant shape parameter. This is a typical example in which the two approaches have system matrices with approximately the same condition number, but the variable shape approach is several decimal places more accurate.
- **centroCenters.m** Produces the centers in the right image of **figure 4**.
- **complexCentroCenters.m** Constructs a centro center distribution on a complexly shaped domain using quasi-random Hammersley points which are placed denser near the boundary than in the interior. Before the centers are extended centrosymmetrically, the domain needs to be rotated so that it is symmetric with respect to the x-axis.
- **interp3d.m** Gaussian RBF interpolation on the surface of a sphere.
- **interp3dCentro.m** Gaussian RBF interpolation on the surface of a sphere as in **interp3d.m** but with a centrosymmetric center distribution. The system matrix as well as all order differentiation matrices will have a centro structure. The centrosymmetric approach executes in approximately half the time that the standard approach takes.
- **condVaccuracy.m** Uses a 1d interpolation problem and the IQ RBF to illustrate the trade off between conditioning and accuracy using both double and extended precision. The script produces the images in **figure 1**.
- **mdiRegularization.m** Interpolates the Franke function on scattered centers located in a domain that is one-fourth of a circle. Compares the accuracy and condition number of the system matrix over a range of shape parameter with and without regularization by the method of diagonal increments. The output is shown in **figure 2**.
- **mdiExample.m** 1d interpolation problem using extended precision and regularization by the method of diagonal increments.
- **rbfInterpConvergence.m** Convergence rate of a RBF interpolant with a fixed shape parameter and increasing N (decreasing distance between centers). The convergence is geometric (also called spectral or exponential) as long as the floating point system can handle the condition number of the system matrix.

- ***rbfInterpConvergenceB.m*** Similar to `rbfInterpConvergence.m` except that the number of centers N is fixed and the shape parameter is decreasing. The convergence is exponential as long as the floating point system can handle the condition number of the system matrix.
- ***poissonCentro.m*** The script solves a 2d steady PDE problem, the Poisson equation $u_{xx} + u_{yy} = -\pi^2 \sin(\pi x) \sin(\pi y)$, on a circular domain with Dirichlet boundary conditions using Kansa's asymmetric RBF collocation method [9]. The asymmetric collocation method discretizes the boundary value problem as $Ha = f$ where H is a $N \times N$ matrix that discretizes the PDE at interior centers and applies boundary conditions at centers located on the boundary. After the linear system is solved for the expansion coefficients, a , the approximate solution is found by a matrix-vector multiplication $u \approx Ba$ where B is the $N \times N$ system matrix. The details of the application of the asymmetric collocation method to steady linear PDEs can be found in reference [21].

The problem is solved two ways: 1) standard algorithms, 2) centrosymmetric algorithms. With $N = 5000$ the accuracy of the two approaches is the same but the centrosymmetric approach is approximately five times faster and requires only half the storage compared to the standard approach.

- ***diffusionReactionCentro.m*** and ***diffusionReactionCentroDriver.m*** The scripts solve the diffusion-reaction PDE $u_t = \nu(u_{xx} + u_{yy}) + \lambda u_2(1 - u)$ on a circular domain with Dirichlet boundary conditions prescribed from the exact solution. The space derivatives in the PDE are discretized by the IQ RBF method and are evaluated by the matrix-vector product Du where D is a differentiation matrix that discretizes the 2d Laplacian operator. The PDE is advanced in time with a fourth-order Runge-Kutta method. The details of the application of the RBF collocation method to time-dependent PDEs can be found in reference [21].

The problem is solved two ways: 1) standard algorithms, 2) centrosymmetric algorithms. With $N = 5000$ the accuracy of the two approaches is the same but the centrosymmetric approach is approximately eight times faster and requires half the storage.

The following test scripts come with the toolbox:

- ***centroCondTest.m*** Verifies the centrosymmetric 2-norm condition number algorithm against the standard algorithm. The two algorithms agree until the matrix becomes very ill-conditioned. As expected there is a slight variation when $\kappa(B) > \mathcal{O}(10^{16})$.
- ***centroSolveAccuracy.m*** Compares the accuracy of the centrosymmetric and standard algorithms for solving a linear system. The linear system is the system (6) for the RBF expansion coefficients over a range of the shape parameter. The centrosymmetric algorithm is slightly more

accurate at most shape parameters and several decimal places more accurate for several shape parameters.

- ***isCentroTest.m*** Depending on how the centers were extended to be symmetric, RBF differentiation matrices will have a (skew) centrosymmetric structure. Reference [18] can be consulted for details.
- ***rbfDerivativeTest.m*** Verifies the accuracy of all derivative approximation methods of the `iqx` and `gax` classes using both double and quadruple precision.

The following benchmark scripts come with the toolbox:

- ***systemSolveBench.m*** Compares the evaluation times of centrosymmetric versus standard algorithms for the solution of a centrosymmetric linear system. The centrosymmetric linear system solving algorithm is faster than the standard algorithm for $N > 350$ and nearly four times as fast for large N .
- ***condBench.m*** Compares the execution times of the centrosymmetric and standard algorithm for calculating the 2-norm condition number of a centrosymmetric matrix. For $N > 100$ the centrosymmetric algorithm is more efficient and for large $N > 2000$ it is nearly five times as fast.
- ***dmFormBench.m*** Compares the execution time of the centrosymmetric and standard algorithm for RBF derivative matrix formation. For larger N , the centrosymmetric algorithm is over three times faster.
- ***multiplicationBench.m*** Compares the execution times of centrosymmetric matrix multiplication to standard matrix multiplication. For $N \geq 900$ the centrosymmetric algorithm is faster and the limiting efficiency factor of two is being approached.
- ***interpBenchExtended.m*** Compares the execution time of a 2d scattered data interpolation problem over a range of the shape parameter using both double and quadruple precision.
- ***centroExtendedConditionNumberBench.m*** Compares the execution times of centrosymmetric and standard algorithms for the 2-norm condition number using both double and quadruple precision. The condition number of a 2000×2000 system matrix is calculated using both double precision and quadruple precision. The centrosymmetric algorithm speeds up the double precision calculation by a factor of approximately 3.8 and the quadruple precision calculation by a factor of 3.3.

Additional results demonstrating the benefits of using extending precision with RBF methods and the efficiency gains from exploiting centrosymmetry can be found in references [20] and [18] for which the MRBFT was used extensively to produce the examples and results.

Summary

The MRBFT is a freely available collection of Matlab files and scripts that implement RBF methods for scattered data interpolation and for the numerical solution of PDEs.

The toolbox has been developed over a period of several years while it has been used in the author's research. The class `rbfx` implements routines for common tasks associated with all RBFs and defines abstract methods for differential operators that must be implemented by all subclasses that define particular RBFs. The class `rbfCenters` implements methods for locating uniform and quasi-random centers in rectangular and circular domains. The class `rbfCentro` implements methods for locating centers in symmetric domains that cause RBF matrices to have a structure that allows for substantial saving in storage requirements and reductions in flop counts for associated linear algebra routines. All MRBFT functions can be implemented using extended precision floating point arithmetic provided that the MCT [1] is installed. Scripts that provide examples, tests, and benchmarks of the MRBFT are included with the distribution. Comments in the class files provide additional documentation.

The author uses the MRBFT in his own research and as a result the toolbox is constantly being improved and new features are being added. Future improvements to the MRBFT include: developing a class of methods for working with the local RBF method, implementing Mercer's method for the GA RBF [3], developing a class that implements rational RBF methods, and adding methods to the `rbfCenters` class that distribute centers in more complexly shaped domains. Updates, bug fixes, and other improvements to the MRBFT are posted at [19] where the project is hosted.

Implementation and architecture

A summary of the implementation of the MRBFT is as follows. The toolbox is implemented in Matlab which is widely used in Mathematics and other scientific disciplines. An object oriented approach is used to organize the functionality of the toolbox.

Quality control

The MRBFT has been developed over a period of several years as it has been used in the author's research. It was used extensively in the preparation of references [20] and [18] which are accompanied by scripts that use the MRBFT to produce the results in the manuscripts. Additionally, as discussed in the overview section, the toolbox comes with a collection of scripts that demonstrate its usage, benchmark its performance, and verify that its algorithms produce the correct results.

(2) Availability

Operating system

The MRBFT is programmed in Matlab which is available on Windows, OS X, and Linux.

Programming language

The MRBFT was developed and tested using Matlab versions 2015b through 2016b [12]. It is not clear as to which version of Matlab first featured object oriented programming (OOP). However, version R2008a made major

changes in the way Matlab implements OOP. Thus, while not tested, it is possible that the MRBFT is compatible with Matlab versions as old as 2008a.

Dependencies

The MRBFT uses the Multiprecision Computing Toolbox for Matlab (MCT) [1] for its extended precision functionality. Note that the MRBFT is in no way dependent on the MCT. The installation of the MCT is not necessary. However, without the MCT, the MRBFT is limited to double precision floating point arithmetic.

List of contributors

The author is the only contributor to the software. However, the contributions of others are welcome.

Software location

Code repository GitHub

Name: scottsarraf/Matlab-RBF-Toolbox

Persistent identifier: <https://doi.org/10.5281/zenodo.221368>

Licence: GNU GPL V3

Date published: 28/03/16

(3) Reuse potential

The potential for others to use the MRBFT is substantial. RBF methods have become very popular in applications. The MRBFT may potentially find use in industrial applications. Research in RBF methods is very active. The MRBFT is a potentially useful tool for faculty research, graduate student research, and undergraduate research projects. The object oriented design of the toolbox makes it extremely easy for it to be extended by subclassing `rbfx` to define new RBFs. The MRBFT can be used to foster the idea of reproducible research in the area of RBF methods. The MRBFT provides core functionality that other research can be built on and for which the code is freely available.

Competing Interests

The author has no competing interests to declare.

References

1. **Advanpix** 2016 Multiprecision computing toolbox for Matlab, version 3.9.9 for 64-bit Linux. <http://www.advanpix.com/>.
2. **Buhmann, M D** 2003 *Radial Basis Functions*. Cambridge University Press, ISBN 0521633389. DOI: <https://doi.org/10.1017/CBO9780511543241>
3. **Fasshauer, G** and **McCourt, M** 2012 Stable evaluation of Gaussian RBF interpolants. *SIAM Journal on Scientific Computing*, 34: 737–762. DOI: <https://doi.org/10.1137/110824784>
4. **Fasshauer, G E** 2007 *Meshfree Approximation Methods with Matlab*. World Scientific. DOI: <https://doi.org/10.1142/6437>
5. **Fornberg, B, Larsson, E** and **Flyer, N** 2011 Stable computations with gaussian radial basis functions in 2-d. *SIAM Journal on Scientific Computing*, 33: 869–892. DOI: <https://doi.org/10.1137/09076756X>

6. **Henderson, H V** and **Searle, S R** 1981 On deriving the inverse of a sum of matrices. *SIAM Review*, 23(2): 53–60. DOI: <https://doi.org/10.1137/1023004>
7. **Huang, C-S**, **Leebn C-F** and **Cheng, A-D** 2007 Error estimate, optimal shape factor, and high precision computation of multiquadric collocation method. *Engineering Analysis with Boundary Elements*, 31: 614–623. DOI: <https://doi.org/10.1016/j.engana-bound.2006.11.011>
8. **Kansa, E J** 1990 Multiquadrics – a scattered data approximation scheme with applications to computational fluid dynamics I: Surface approximations and partial derivative estimates. *Computers and Mathematics with Applications*, 19 (8/9): 127–145. DOI: [https://doi.org/10.1016/0898-1221\(90\)90270-T](https://doi.org/10.1016/0898-1221(90)90270-T)
9. **Kansa, E J** 1990 Multiquadrics – a scattered data approximation scheme with applications to computational fluid dynamics II: Solutions to parabolic, hyperbolic, and elliptic partial differential equations. *Computers and Mathematics with Applications*, 19(8/9): 147–161. DOI: [https://doi.org/10.1016/0898-1221\(90\)90271-K](https://doi.org/10.1016/0898-1221(90)90271-K)
10. **Kansa, E J** and **Carlson, R** 1992 Improved accuracy of multiquadric interpolation using variable shape parameters. *Computers and Mathematics with Applications*, 24: 99–120. DOI: [https://doi.org/10.1016/0898-1221\(92\)90174-G](https://doi.org/10.1016/0898-1221(92)90174-G)
11. **Larsson, E** 2012 A MATLAB implementation of the RBF-QR method. http://www.it.uu.se/research/scicomp/software/rbf_qr.
12. **MATLAB** 2015 *version 8.6 (R2015b)*. The MathWorks Inc., Natick, Massachusetts.
13. **McCourt, M** 2014 GaussQR: Stable Gaussian computation. <http://math.iit.edu/~mccomic/gaussqr/>.
14. **Niederreiter, H** 1992 *Random Number Generation and Quasi-Monte Carlo Methods*. CBMS-NSF, SIAM, Philadelphia, DOI: <https://doi.org/10.1137/1.9781611970081>
15. **Piegorsch, W** and **Casella, G** 1989 The early use of matrix diagonal increments in statistical problems. *SIAM Review*, 31: 428–434. DOI: <https://doi.org/10.1137/1031089>
16. **Sarra, S A** 2011 Radial basis function approximation methods with extended precision floating point arithmetic. *Engineering Analysis with Boundary Elements*, 35: 68–76. DOI: <https://doi.org/10.1016/j.engana-bound.2010.05.011>
17. **Sarra, S A** 2014 Regularized symmetric positive definite matrix factorizations for linear systems arising from RBF interpolation and differentiation. *Engineering Analysis with Boundary Elements*, 44: 76–86. DOI: <https://doi.org/10.1016/j.engana-bound.2014.04.019>
18. **Sarra, S A** 2016 Radial basis function methods – the case of symmetric domains. *Under review, Numerical Methods for Partial Differential Equations*.
19. **Sarra, S A** 2016 A Matlab radial basis function toolbox with symmetry exploitation, regularization, and extended precision. <http://www.scottsarra.org/rbf/rbf.html>.
20. **Sarra, S A** and **Cogar, S** 2017 An examination of evaluation algorithms for the RBF method. *Engineering Analysis with Boundary Elements*, 17: 36–45. DOI: <https://doi.org/10.1016/j.engana-bound.2016.11.006>
21. **Sarra, S A** and **Kansa, E J** 2009 Multiquadric radial basis function approximation methods for the numerical solution of partial differential equations. *Advances in Computational Mechanics*, 2.
22. **Sarra, S A** and **Sturgill, D** 2009 A random variable shape parameter strategy for radial basis function approximation methods. *Engineering Analysis with Boundary Elements*, 33: 1239–1245. DOI: <https://doi.org/10.1016/j.engana-bound.2009.07.003>
23. **Wendland, H** 2005 *Scattered Data Approximation*. Cambridge University Press, DOI: <https://doi.org/10.1017/CBO9780511617539>
24. **Wong, T**, **Luk, W** and **Heng, P** 1997 Sampling with Hammersley and Halton points. *Journal of Graphics Tools*, 2: 9–24. DOI: <https://doi.org/10.1080/10867651.1997.10487471>

How to cite this article: Sarra, S A 2017 The Matlab Radial Basis Function Toolbox. *Journal of Open Research Software*, 5: 8, DOI: <https://doi.org/10.5334/jors.131>

Submitted: 28 May 2016 **Accepted:** 13 January 2017 **Published:** 27 March 2017

Copyright: © 2017 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <http://creativecommons.org/licenses/by/4.0/>.